

FIG. 1

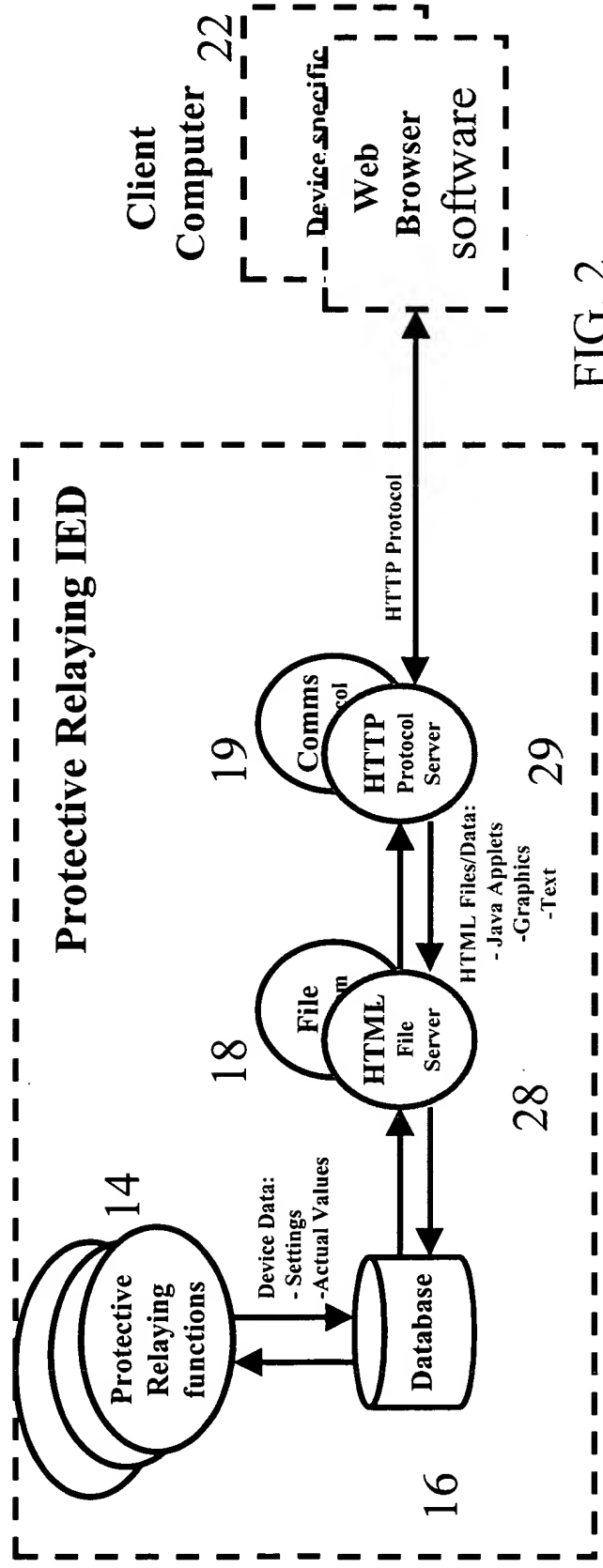


FIG. 2

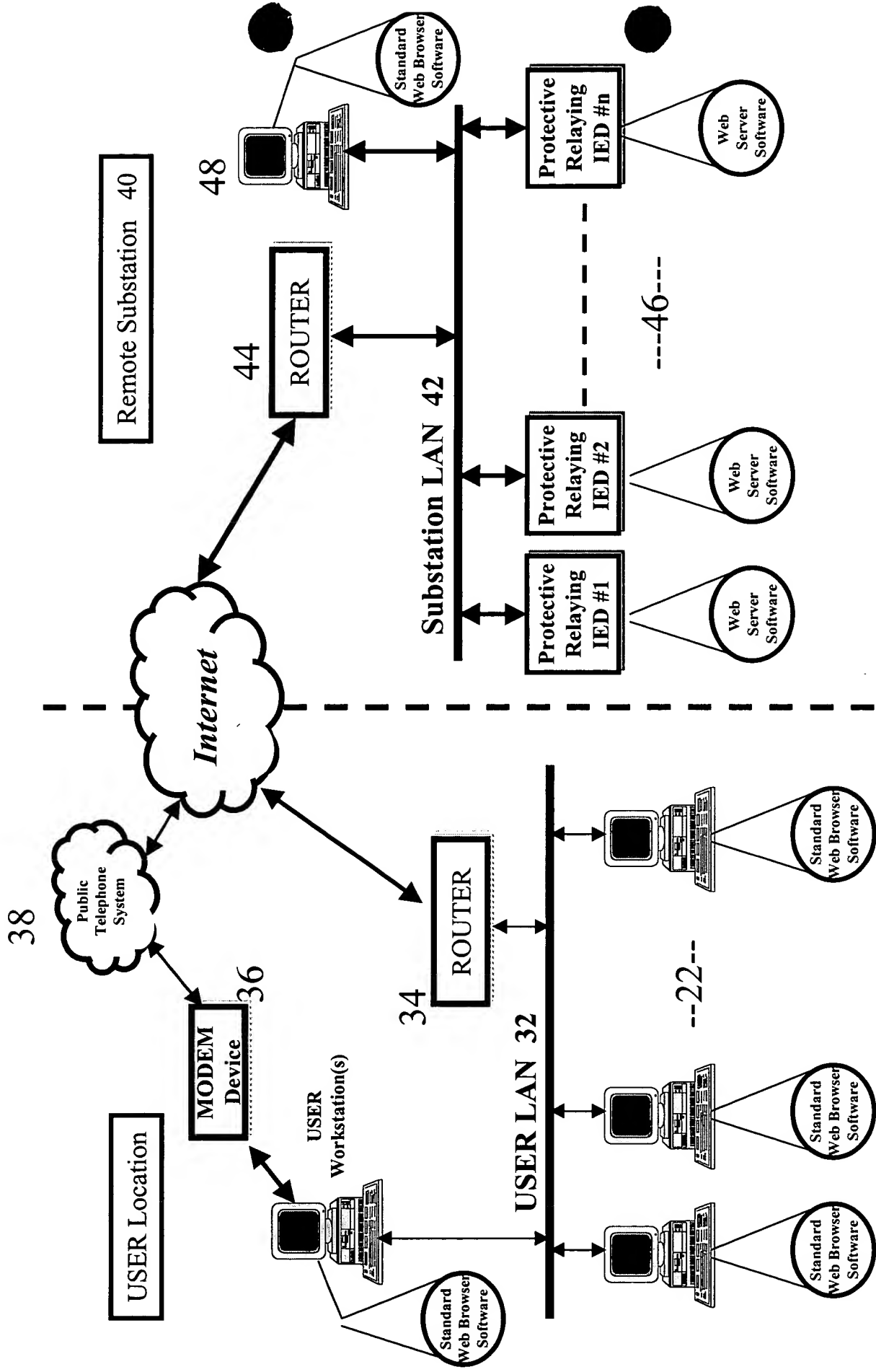


FIG. 3

UR HTTP Server Protocol Stack

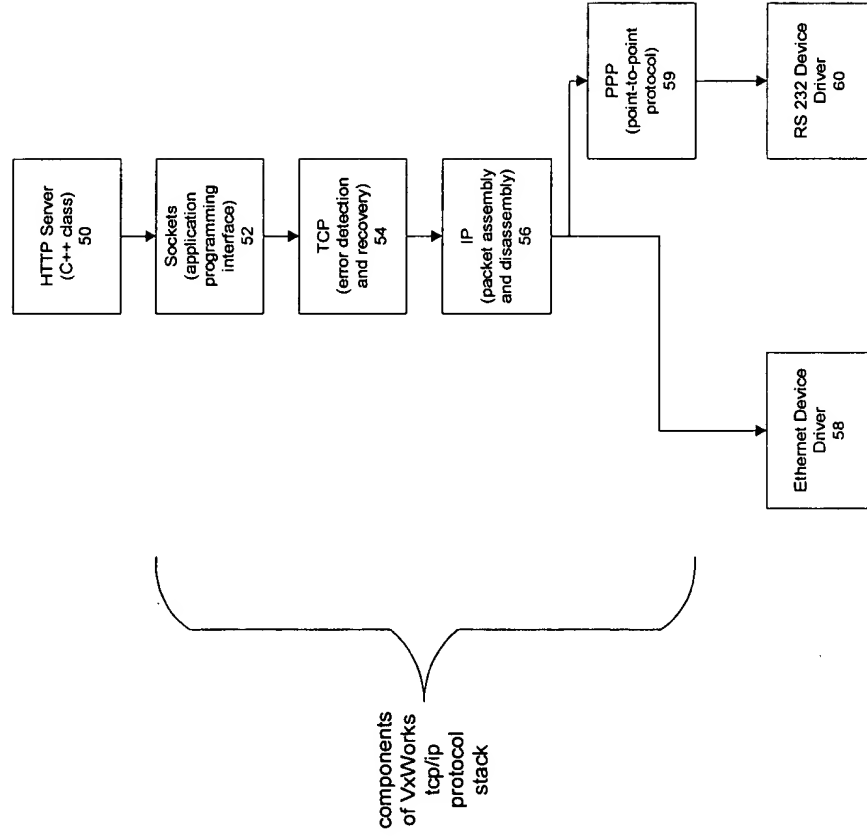


FIG. 4

UR HTTP Server Data Flow

50

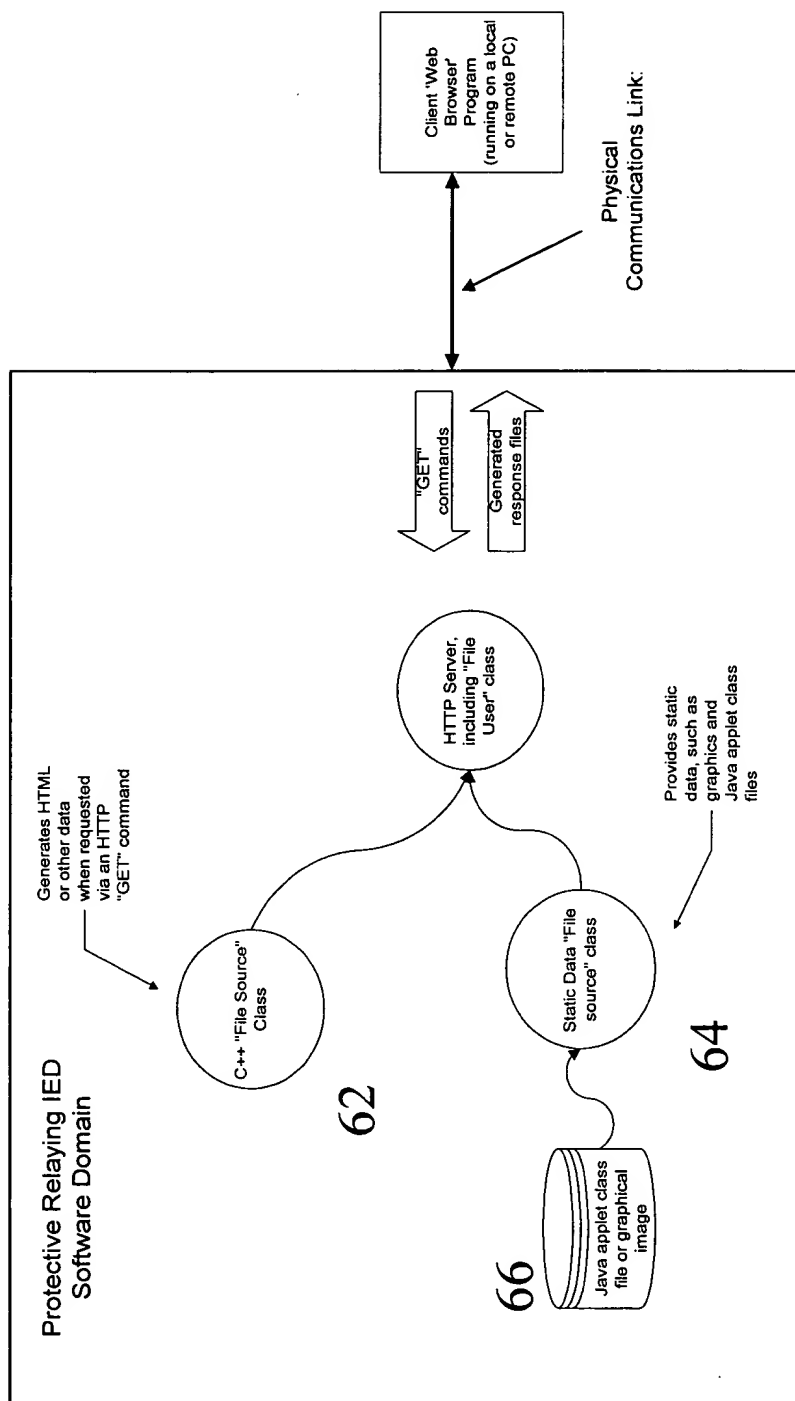


FIG. 5

**TITLE: PROTECTIVE RELAY WITH
EMBEDDED WEB SERVER**

FIG. 6 Source Code

004290" 01050960

Listing 1: COM_Webserver.h.....	1
Listing 2: COM_Webserver.cpp.....	2
Listing 3: UTL_FileSource.h.....	31
Listing 4: UTL_FileSource.cpp.....	31
Listing 5: UTL_WebPage.h.....	32
Listing 6: UTL_WebPage.cpp.....	36
Listing 7: UTL_FileUser.h.....	38
Listing 8: UTL_FileUser.cpp.....	50

Listing 1: COM_Webserver.h

```

/*****
 * Copyright (C) General Electric Co.   GE Confidential and Proprietary
 *
 * DESCRIPTION  This file contains the Modbus/TCP communications port sub-class.
 *
 *****/

#ifndef COM_WebServer_H
#define COM_WebServer_H

// INCLUDES
#include "COM_Hardware.h"
#include "SYS_DPRAM.h"
#include <assert.h>
#include "DB_NotificationSource.h"

#define MAX_HTTP_CONNECTIONS      3           // maximum number of simultaneous connections

class UTL_lmsTimer;

////////////////////////////////////
//
// Web server task -- handles HTTP protocol connections over TCP/IP, so you
// can access any ethernet-capable relay with a web browser.  It uses
// the "default.htm" menu (UTL_WebMenu class) as the main entry point to the
// web site.  Other UTL_WebPage objects may be accessed through the menu structure
// or by specifying the URL for each page.  The UTL_WebPage objects are distributed
// through the UR firmware files, so each one is near the data it requires.
// You can also read non-webpage UTL_FileSource files with the web browser, but
// they don't appear in menus.
// <BR>
// For more information about web pages and other files, see UTL_WebPage and
// UTL_FileUser.
//
////////////////////////////////////
class COM_WebServer : public DB_NotificationSource
{
public:
    COM_WebServer();                               // Constructor
    virtual ~COM_WebServer();                       // Destructor
    virtual void sendFrame(unsigned char *buffer, UR_UINT16 length, int con_sFd );
    void connect_Task(void);
    static int call_connect_Task(COM_WebServer *);
    static int call_read_Task( COM_WebServer * obj, int connectionNumber );
    void read_Task( int connectionNumber );
    void acceptNotification(DB_NotificationSource *source, int param);
    // Get the number of active connections
    int getConnectionCount(void) { return connectionCount; }

    // Returns a pointer to the only object of this class.
    static COM_WebServer * const find( void )
    {
        assert( the_COM_WebServer );
        return the_COM_WebServer;
    }
}

```

002230 "062700


```

};

#include "COM_ModbusAddress.h"
class DB_MemoryMapWebPage : public UTL_WebPage
{
public:
    DB_MemoryMapWebPage(const char*filename)
        : UTL_WebPage(filename)
    {
        menuFileName = "default.htm";
    }
protected:
    virtual void getBody(UTL_FileUser & dest,int optionCount, const char *options[],const
char *filename)
    {
        (void)filename;
        UR_MODULE lastModule=UR_MODULE(-1), selectedModule=UR_MODULE(-1);
        UR_BOOLEAN summary = UR_FALSE;

        if( optionCount )
        {
            int m = 0;
            (void)sscanf(options[0],"%d",&m);
            selectedModule = (UR_MODULE)m;
        }
        else
            summary = UR_TRUE; // if no module specified, present a summary

        UTL_WebPage::Table t(3,dest);
        if( summary )
        {
            t.startBannerCell();
            dest.puts( "MEMORY MAP SUMMARY");

            t.startHeadingCell();
            dest.puts("Address");
            t.startHeadingCell("left");
            dest.puts("Module");
            t.startHeadingCell();
            dest.puts("Array Size");
        }
        else
        {
            t.startTable(7);
            t.startBannerCell();
            dest.printf( "MEMORY MAP FOR \"%s\"",
                        SYS_Product::find()->getName(selectedModule) );

            t.startHeadingCell();
            dest.puts("Address");
            t.startHeadingCell("left");
            dest.puts("Name");
            t.startHeadingCell();
            dest.puts("Type");
            t.startHeadingCell();
            dest.puts("Min");
            t.startHeadingCell();
            dest.puts("Max");
            t.startHeadingCell();
            dest.puts("Value");
            t.startHeadingCell();
            dest.puts("Unit");
        }
        DB_DataItem * d = 0;
        UR_UINT16 moduleIndex = 0;
        UR_UINT16 arrayIndex = 0;
        UR_UINT16 nextAddress = 0xFFFF; // address of next array element
        UR_UINT16 addr=0;
        do
        {
            COM_ModbusAddress * ma = COM_ModbusAddress::find(addr);
            if( ma )
            {
                if(
                    addr==nextAddress // possibly next array index for current item
                    || ma->getItem() != d // new data item

```

00.239.0.050960


```

current item      || ma->getModuleIndex() != moduleIndex // next module index for
                  )
                  {
                    if( ma->getItem() != d )
                    {
                      arrayIndex = 0;
                      d = ma->getItem();
                      moduleIndex = ma->getModuleIndex();
                    }
                    else if( ma->getModuleIndex() != moduleIndex )
                    {
                      arrayIndex = 0; // reset array index at start of new module
                      moduleIndex = ma->getModuleIndex();
                    }
                    else
                      arrayIndex++; // traversing item array in same module of same
item
                  if( summary )
                  {
                    if( d->module != lastModule )
                    {
                      t.startCell();
                      dest.printf("%04X",addr);
                      t.startCell("left");
                      dest.printf("<A HREF=%s?%d>%s</A>",
                                getFileName(),
                                (int)d->module,
                                SYS_Product::find()->getName(d->module)
                                );
                      t.startCell();
                      dest.printf("%d", (int)SYS_Product::find()->getSize(d-
>module));
                      lastModule = d->module;
                    }
                    else if( d->module == selectedModule )
                    {
                      nextAddress = addr + (d->getSize()+1)/2; // here's where the
next array element is
                      char s[100];
                      char a[100];
                      UR_UINT16 c;
                      t.startCell();
                      dest.printf("%04X",addr);
                      t.startCell("left");
                      d->getFormattedName(UR_TRUE,&c,&s[0],moduleIndex,arrayIndex);
                      dest.puts(&s[0]);
                      t.startCell();
                      if( d->attrib.eeprom )
                        dest.puts("Read/Write Setting");
                      else if( d->attrib.write )
                        dest.puts("Writable Actual");
                      else if( d->attrib.sram )
                        dest.puts("Non-volatile Actual");
                      else
                        dest.puts("Read Only");
                      t.startCell();
                      (void)d->getMinimum(&s[0]);
                      (void)d->toAscii(&c,a,s,moduleIndex,arrayIndex,0);
                      if( !*a )
                      {
                        t.setFontBold();
                        dest.puts("(?)");
                      }
                      else
                        dest.puts(a);
                      t.startCell();
                      (void)d->getMaximum(&s[0]);
                      (void)d->toAscii(&c,a,s,moduleIndex,arrayIndex,0);
                      if( !*a )
                      {
                        t.setFontBold();
                        dest.puts("(?)");
                      }

```

09605010-062700

```

        }
        else
            dest.puts(a);
        t.startCell();
        (void)d->get(s,moduleIndex,arrayIndex,0);
        (void)d->toAscii(&c,a,s,moduleIndex,arrayIndex,0);
        if( !*a )
        {
            t.setFontBold();
            dest.puts("(?)");
        }
        else
            dest.puts(a);
        t.startCell();
        if( ! * webString(a,d->getUnit(moduleIndex,arrayIndex,0)) )
            strcpy(a,"&nbsp;");
        dest.puts(a);
    }
}

} while( ++addr );

}
virtual void printTitle(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename)
{
    (void)optionCount;
    (void)options;
    (void)filename;
    dest.puts( "Modbus Memory Map");
}

};

#include "memLib.h"
#include "DSP_Card.h"
#include "UTL_TaskDataPointer.h" // for testing only
class WEB_MiscStats : public UTL_WebPage
{
public:
    WEB_MiscStats(const char*filename) : UTL_WebPage(filename), tdp(30)
    {
        menuFileName = "DiagnosticsMenu.htm";
    }
protected:
    virtual void getBody(UTL_FileUser & dest,int optionCount, const char *options[],const
char *filename)
    {
        (void)optionCount;
        (void)options;
        (void)filename;

        UTL_WebPage::Table t(2,dest);

        for( int i=0; i<N_DSPS; i++ )
        {
            DSP_Card * d = DSP_Card::find(i);
            if( d )
            {
                t.startCell("right");
                t.setFontBold();
                dest.printf("DSP %d usage:",i);
                const DSP_State & p = d->getDspState();
                t.startCell();
                dest.printf("%.1f%%",float(p.Dsp_Usage)/10.0);
            }
            t.startCell("right");
            t.setFontBold();
            dest.printf("Largest Free Memory Block");
            t.startCell();
            dest.printf( "%d bytes",memFindMax());

            static int myNumber=1;
            char * myName = (char*)(tdp.get());
            if( !*myName )
                (void)sprintf(&myName[0],"TASK %d",myNumber++);
            t.startCell("right");

```

09605010 062700

```

        t.setFontBold();
        dest.puts("HTTP Connection Number:");
        t.startCell();
        dest.puts(myName);
    }
    virtual void printTitle(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename)
    {
        (void)optionCount;
        (void)options;
        (void)filename;
        dest.puts( "Miscellaneous Diagnostics" );
    }
    UTL_TaskDataPointer tdp;    // for testing only
};

//*****
//*****

#define SOCKET_ERROR ERROR

#define INTEGER_2    2        // the number 2
#define INTEGER_3    3        // the number 3
#define INTEGER_4    4        // the number 4

// The following definitions allow incorporation of socket calls into both the
// GNU and WIN32 builds.
#ifdef WIN32
    // WIN32 version of socket calls.

    extern void      SOCKET_CALL_INET_NTOA_B(unsigned long inetAddress, char *pString);
    static void      SOCKET_CALL_INET_NTOA_B(in_addr inetAddress, char *pString)
    {
        SOCKET_CALL_INET_NTOA_B(inetAddress.s_addr, pString);
    }
    extern STATUS    SOCKET_CALL_SETSOCKOPT (int s, int level, int optname, char *optval,
int optlen);
    extern int       SOCKET_CALL_SEND (int s, char *buf, int buflen, int flags);
    extern int       SOCKET_CALL_ACCEPT (int s, struct sockaddr *addr, int *addrlen);
    extern STATUS    SOCKET_CALL_LISTEN (int s, int backlog);
    extern STATUS    SOCKET_CALL_BIND (int s, struct sockaddr *name, int namelen);
    extern int       SOCKET_CALL_SOCKET (int domain, int type, int protocol);
    extern int       SOCKET_CALL_RECV (int s, char *buf, int buflen, int flags);
    extern STATUS    SOCKET_CALL_CLOSE (int fd);
    extern STATUS    SOCKET_CALL_SHUTDOWN(int s, int how);
    extern STATUS    SOCKET_CALL_CONNECT(int s, struct sockaddr * name, int namelen);
#else
    // GNU version of socket calls -- simply map them to the VxWorks function names.

#define SOCKET_CALL_INET_NTOA_B inet_ntoa_b
#define SOCKET_CALL_SETSOCKOPT setsockopt
#define SOCKET_CALL_SEND send
#define SOCKET_CALL_ACCEPT accept
#define SOCKET_CALL_LISTEN listen
#define SOCKET_CALL_BIND bind
#define SOCKET_CALL_SOCKET socket
#define SOCKET_CALL_RECV recv
#define SOCKET_CALL_CLOSE close
#define SOCKET_CALL_SHUTDOWN shutdown
#define SOCKET_CALL_CONNECT connect

#endif

COM_WebServer * COM_WebServer::the_COM_WebServer = 0;

////////////////////////////////////
//
// FUNCTION      COM_WebServer::COM_WebServer
//
// DESCRIPTION    TcpPort class constructor.
//
////////////////////////////////////
COM_WebServer::COM_WebServer(void)
{
    the_COM_WebServer = this;

```

00/02/2000 10:50:10

```

isInitialized = UR_FALSE;
pleaseKillMe = false;
connectionCount = 0;
numRunningTasks = 0;

for( int i=0; i<MAX_HTTP_CONNECTIONS; i++ )
    connectionTimers[i] = new UTL_lmsTimer(SOCKET_TIMEOUT * 1000); // convert to
milliseconds
IP_Address.registerForNotification(this);

// Create our web pages
(void)new UTL_StaticFile("bug.gif", (unsigned char*)&GifBug, sizeof(GifBug));
(void)new UTL_StaticFile("UR_Grid.class", (unsigned char*)&UR_GridClass,
sizeof(UR_GridClass));
(void)new DB_MemoryMapWebPage("memoryMap.htm");
(void)new UTL_WebMenu("DeviceInfoMenu.htm", "default.htm", "Device Information Menu");
(void)new UTL_WebMenu("DiagnosticsMenu.htm", "default.htm", "Diagnostics
Menu", FACTORY_LEVEL);
(void)new WEB_MiscStats("MiscStats.htm");
(void)new WEB_CustomerSupport("CustomerSupport.htm");
}

////////////////////////////////////
//
// FUNCTION      COM_WebServer::~COM_WebServer
//
// DESCRIPTION   COM_WebServer class destructor.
//
////////////////////////////////////
COM_WebServer::~COM_WebServer()
{
    int i;

    pleaseKillMe = true;           // let tasks know we're all done

    // Kill all the connected sockets
    for( i=0; i<MAX_HTTP_CONNECTIONS; i++ )
    {
        connectionTimers[i]->stop();
        connectionTimers[i]->setTimeDelay(1);
        connectionTimers[i]->start();           // this should cause existing connections
to die
    }

    // Kill the unconnected sockets
    for( i=0; i<MAX_HTTP_CONNECTIONS; i++ )
    {
        // Connect to each socket, then disconnect -- the receive task will see the
        // pleaseKillMe flag, and quit.

        // socket structure to use vxWorks TCP-functions without causing a pclint warning
        union socket_stuff
        {
            sockaddr_in in_Addr;
            sockaddr     sock_Addr;
        };

        union socket_stuff server_stuff; // server socket address
        int                 sockAddrSize; // size of socket address structure
        u_long localhost = 0x7f000001; // "localhost" address (127.0.0.1)
        int clientFd; // client socket

        // Create client socket
        clientFd = SOCKET_CALL_SOCKET (AF_INET, SOCK_STREAM, 0);
        if (clientFd == SOCKET_ERROR)
        {
            printf("\nSocket creation error.\n");
            continue;
        }

        // set up the local address
        sockAddrSize = sizeof (server_stuff.in_Addr);
        bzero ( (char *)&server_stuff.in_Addr, sockAddrSize );
        server_stuff.in_Addr.sin_family = AF_INET;
        server_stuff.in_Addr.sin_port = htons (SERVER_PORT_NUM);
    }
}

```

002290"07050600

[illegible]

```

void COM_WebServer::connect_Task()
{
    // socket structure to use vxWorks TCP-functions without causing a pclint warning
    union socket_stuff
    {
        sockaddr_in in_Addr;
        sockaddr sock_Addr;
    };

    union socket_stuff server_stuff;    // server socket address
    int sockAddrSize;    // size of socket address structure
    int ix = 0;    // counter for read task names
    char task_name[16];    // name of read tasks
    int optval;    // socket options

    // set up the local address
    sockAddrSize = sizeof (server_stuff.in_Addr);
    bzero ( (char *)&server_stuff.in_Addr, sockAddrSize );
    server_stuff.in_Addr.sin_family = AF_INET;
    server_stuff.in_Addr.sin_port = htons (SERVER_PORT_NUM);
    server_stuff.in_Addr.sin_addr.s_addr = htonl (INADDR_ANY);

    // create a TCP-based socket
    if ((sFd = SOCKET_CALL_SOCKET (AF_INET, SOCK_STREAM, 0)) == SOCKET_ERROR)
    {
        printf("\nSocket creation error.\n");
        return;
    }

    // set socket options
    // optval = 1; // SO_KEEPAIVE on
    // SOCKET_CALL_SETSOCKOPT (sFd, SOL_SOCKET, SO_KEEPAIVE, (caddr_t) &optval, sizeof
    (optval));

    optval = 1; // TCP_NODELAY on
    SOCKET_CALL_SETSOCKOPT (sFd, SOL_SOCKET, TCP_NODELAY, (caddr_t) &optval, sizeof
    (optval));

    optval = 1; // SO_REUSEADDR on
    SOCKET_CALL_SETSOCKOPT (sFd, SOL_SOCKET, SO_REUSEADDR, (caddr_t) &optval, sizeof
    (optval));

    struct linger lng;
    lng.l_linger = 0;    // zero timeout on linger
    lng.l_onoff = 1;
    SOCKET_CALL_SETSOCKOPT (sFd, SOL_SOCKET, SO_LINGER, (caddr_t) &lng, sizeof (lng));

    // bind socket to local address
    if (SOCKET_CALL_BIND (sFd, &server_stuff.sock_Addr, sockAddrSize) == SOCKET_ERROR)
    {
        printf("\nSocket bind error.\n");
        SOCKET_CALL_CLOSE (sFd);
        return;
    }

    // create queue for client connection requests
    if (SOCKET_CALL_LISTEN (sFd, MAX_HTTP_CONNECTIONS) == SOCKET_ERROR)
    {
        printf("\nSocket listen error.\n");
        SOCKET_CALL_CLOSE (sFd);
        return;
    }

    for( ix=0; ix<MAX_HTTP_CONNECTIONS; ix++ )
    {
        sprintf (task_name, "WebRx%d", ix);
        (void)taskSpawn(task_name, SYS_Application::utilityPriority, 0, 10000,
            (FUNCPTR) call_read_Task, (int) this, ix, 0, 0, 0, 0, 0, 0, 0);
    }

    // Now loop forever checking the timers for all the connections
    int t = sysClkRateGet();    // once per second should do it
    while ( sFd != SOCKET_ERROR )    // keep going until main socket is closed by
    destructor
    {
        taskDelay(t);
    }
}

```

00229007050960

```

for( int i=0; i<MAX_HTTP_CONNECTIONS; i++ )
{
    if( connected_sFd[i] != SOCKET_ERROR && connectionTimers[i]->isElapsed() )
    {
        // timer elapsed - kill the connection, but not in a polite way
#ifdef DEBUG_HTTP
        printf("http %d: timed out -- shutting down\n", i);
#endif
        (void)SOCKET_CALL_SHUTDOWN(connected_sFd[i],2);
    }
    // If we're shutting down and this is the last task, die
    if( pleaseKillMe && numRunningTasks<=1 )
        break;
}
printf("Web server connect task is finished\n");
SOCKET_CALL_CLOSE(sFd);
}

// Create a "page not found" page when the browser has requested a file
// which either doesn't exist or is inaccessible.
void COM_WebServer::notFoundPage(int connected_sFd)
{
    char response[1000];

    sprintf( response,
        "<HTML>\n"
        "<HEAD>"
        "<meta http-equiv=\"refresh\" content=\"5\">"
        "<TITLE>Page Not Found</TITLE></HEAD>\n"
        "<BODY BGCOLOR=\"#FFFFFF\">\n"
        "<H1>PAGE NOT FOUND</H1><BR><BR>\n"
        "<HR><STRONG><A HREF=default.htm>Click Here For The Main Menu</A></STRONG><HR>\n"
        "</BODY></HTML>\n\r\n"
    );
#ifdef _WIN32
    printf("Sending-----\n%s\n-----", response);
#endif
    sendFrame( (unsigned char *)response, strlen(response), connected_sFd );
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// FUNCTION          COM_WebServer::read_Task
//
// DESCRIPTION       Wait for data from a socket and then send it to the attached
//                   protocol application.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void COM_WebServer::read_Task
{
    int                connectionNumber    // connection number -- identifies this task
}
{
    UTL_TaskDataBlock taskDataBlock;      // each task gets a data block for task-specific
values
    char inet_name[18];                   // socket addr buffer for inet_ntoa_b()
    // socket structure to use vxWorks TCP-functions without causing a pclint warning
    union socket_stuff
    {
        sockaddr_in in_Addr;
        sockaddr     sock_Addr;
    };
    union socket_stuff client_stuff;      // client socket address
    int                sockAddrSize = sizeof(client_stuff); // size of socket address
structure
    UTL_WatchDog wd(10000, false);

    // accept new connect requests and spawn tasks to process them
    while ( sFd != SOCKET_ERROR)
    {
        if ( (connected_sFd[connectionNumber] = SOCKET_CALL_ACCEPT (sFd,
&client_stuff.sock_Addr, &sockAddrSize)) == SOCKET_ERROR)
        {
            printf("\nSocket #%d accept error.\n", connectionNumber);
            SOCKET_CALL_CLOSE (sFd);
        }
    }
}

```

```

        break;
    }
    // Shut down if asked to do so
    else if( pleaseKillMe )
    {
        SOCKET_CALL_CLOSE(connected_sFd[connectionNumber]);
        break;
    }
    wd.kick();
    struct linger lng;
    lng.l_linger = 0;          // zero timeout on linger
    lng.l_onoff = 1;
    SOCKET_CALL_SETSOCKOPT (connected_sFd[connectionNumber], SOL_SOCKET, SO_LINGER,
(caddr_t) &lng, sizeof (lng));

    // Start the dead connection timer
    connectionTimers[connectionNumber]->start();
    // convert the client address to internet address form

    SOCKET_CALL_INET_NTOA_B( client_stuff.in_Addr.sin_addr, inet_name );
    connectionCount++;
#ifdef DEBUG_HTTP
    printf ("\nSocket # %d open. Connection count = %d\n", connectionNumber,
connectionCount);
#endif

    unsigned char    clientRequest[1200]; // request/message from client
    int              nRead;              // number of bytes read

    // read client request and process messages.
    while( (nRead = SOCKET_CALL_RECV( connected_sFd[connectionNumber],
                                     (char*)clientRequest,
                                     sizeof(clientRequest)-1,
                                     0)) > 0 )
    {
        wd.kick();
        // Shut down if asked to do so
        if( pleaseKillMe )
            break;

        // Got something from the client -- process it.
        clientRequest[nRead] = 0; // null terminate

        // re-start the dead connection timer
        connectionTimers[connectionNumber]->start();

#ifdef DEBUG_HTTP>1
        // message display (enable for debugging if required)
        printf ("\nMESSAGE FROM CLIENT on # %d (Internet Address %s, port %d, length
%d, sFd %d)\n",
connectionNumber, address, port, nRead, connected_sFd[connectionNumber]);
        for (int i=0; i<nRead; i++)
            printf("%u ", clientRequest[i]);
        printf("\n");
#endif

#ifdef _WIN32
        printf("HTTTPD # %d GOT ::::::::::::::\n%s\n-----\n",
connectionNumber, (char*)clientRequest );
#endif

        char fileNameToGet[500] = "/";

        if( ! strcmp((char*)clientRequest, "GET", 3) )
        {
            // check the client's authorization
            clientPassword[0] = 0; // kill off the existing password information
            char *p = (char*)clientRequest;
            char *line = p;
            char *tokens[20];
            int tokenNumber = 0;
            tokens[0] = p;
            do
            {
                switch( *p )
                {
                    case '\r':

```

002230-062700 09605010-062700


```

        f->get(u,optionCount,optionArray,fileNameToGet);
        u.flush();
    }
    else
        notFoundPage(connection_sFd[connectionNumber]);

    #if DEBUG_HTTP
        printf("http %d: transmission complete\n", connectionNumber);
    #endif

    break;
}

if (nRead == SOCKET_ERROR) // error from read()
    printf ("\nSocket #d read error.\n", connectionNumber);

// Stop the dead connection timer, since we have come out cleanly
connectionTimers[connectionNumber]->stop();
// interlock to avoid fight with timer
int savedFd = connection_sFd[connectionNumber];

STATUS err;
if( savedFd != SOCKET_ERROR )
{
    #if DEBUG_HTTP
        printf("http %d: normal shutdown", connectionNumber);
    #endif

    err = SOCKET_CALL_SHUTDOWN(savedFd,1); // shut down send side
    if( err == SOCKET_ERROR )
        printf("\nhhttp %d: shutdown error\n", connectionNumber);
    // make sure all the reads are finished
    // (seems to be necessary to clean out the final ACK)
    while( SOCKET_CALL_RECV( savedFd, (char*)clientRequest,
        sizeof(clientRequest)-1, 0) > 0 )
    {
        // just loop
        wd.kick(); // kick again to re-start the watchdog
    }
    #if DEBUG_HTTP
        printf(".");
    #endif

    (void)SOCKET_CALL_SHUTDOWN(savedFd,2); // shut down everything
    #if DEBUG_HTTP
        printf("close...");
    #endif

    connected_sFd[connectionNumber] = SOCKET_ERROR; // mark the socket closed so
the timer can't mess us up
    err = SOCKET_CALL_CLOSE (savedFd); // close server socket connection
    if( err == SOCKET_ERROR )
        printf("\nhhttp %d: close error\n", connectionNumber);
}

    #if DEBUG_HTTP
        printf ("\nSocket ##d closed.\n", connectionNumber);
    #endif
    connectionCount--;
}

}

////////////////////////////////////
//
// First time in, starts the web connect task
//
////////////////////////////////////
void COM_WebServer::acceptNotification
(
    DB_NotificationSource *source, // not used
    int param // not used
)
{
    (void) source; // use this parameter to avoid a compiler warning
    (void) param; // use this parameter to avoid a compiler warning

    if (isInitialized == UR_FALSE)
    {

```

002290"062700 09605010


```

UTL_FileSource( const char *filename, DB_SECURITY_LEVEL anAccessLevel=NO_LEVEL);
virtual ~UTL_FileSource();
// Overridable function gets the file into a file user, by calling the
// write and puts functions in the UTL_FileUser.
virtual void get(
    UTL_FileUser & dest,        // put output here
    int optionCount,           // number of options
    const char *options[],      // options, if any
    const char *filename        // filename being got, in case the filename contains
options
) = 0;
virtual UR_BOOLEAN isOne( const char * filename );
// Get the file name
const char * getFileName(void) { return theFileName; }
// Get a pointer to the first file
static UTL_FileSource * getFirst(void) { return head; }
// Get a pointer to the next file
UTL_FileSource * getNext(void) { return next; }

UR_BOOLEAN isAccessible(void);

    // Find the object corresponding to the given filename
    static UTL_FileSource * find(const char *filename);

static void deleteAll(void);

    virtual void printTitle(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);

    // access function for menu file name
    const char * getMenuFileName(void) { return menuFileName; }
protected:

    UR_BOOLEAN filenameCharsMatch(char c1, char c2);        // compare two filename characters
    UR_BOOLEAN filenameMatch( const char * basename, const char * filename, unsigned &
number );

    char * theFileName;
    UTL_FileSource * next;                                // pointer to next instance
    static UTL_FileSource * head; // pointer to first instance
    const char * menuFileName;                            // include this file in the named menu file
    DB_SECURITY_LEVEL accessLevel; // access level required to read this file
};

#endif

```

Listing 4: UTL_FileSource.cpp

```
#include "UTL_FileSource.h"
#include "UTL_FileUser.h"
#include "MMI_Application.h"
#include <assert.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
```

```

        UTL_FileUser & dest,      // destination object
        int optionCount,
        const char *options[],
        const char *filename
    )
    {
        (void)optionCount;
        (void)options;
        (void)filename;
        dest.puts("A directory of all the files in the relay");
    }

    virtual void get(
        UTL_FileUser & dest,      // put output here
        int optionCount,          // number of options
        const char *options[],    // options, if any
        const char *filename      // filename being got, in case the filename contains
options
    )
    {
        (void)optionCount;
        (void)options;
        (void)filename;
        dest.puts("File directory\r\n\r\n");
        UTL_FileSource * it = UTL_FileSource::getFirst();
        while( it )
        {
            if( it->isAccessible() )
            {
                const char *dummyOptions[1];
                dest.printf("%-50.200s: ", it->getFileName());
                it->printTitle(dest,0,dummyOptions,it->getFileName());
                dest.puts("\r\n");
            }
            it = it->getNext();
        }
    }
};

//-----
// Constructor -- record the file info.
// If your filename has a leading slash or backslash, it gets removed, so that
// all files are relative to the root ("/") directory.
UTL_FileSource::UTL_FileSource(
        const char *filename,          // file name
        DB_SECURITY_LEVEL anAccessLevel // access level
    )
    {
        // insert into linked list
        next = head;
        head = this;
        assert( filename );
        if( *filename == '\\' || *filename == '/' )
            filename++; // skip over leading slash or backslash, if supplied
        theFileName = new char[strlen(filename)+1];
        assert(theFileName);
        strcpy(theFileName,filename);
        menuFileName = ""; // subclasses may override as required
        accessLevel = anAccessLevel;
        if( !directoryCreated )
        {
            directoryCreated = UR_TRUE;
            (void)new UTL_FileSourceDir("DIR.TXT");
        }
    }

//-----
// destructor
UTL_FileSource::~UTL_FileSource()
{
    // delete stuff
    delete[] theFileName;
    // unlink from list
    if( head == this )
        head = this->next;
    else
    {

```

09605010-062700

```

        UTL_FileSource * it = head;

        while( it )
        {
            if( it->next == this )
            {
                it->next = this->next;
                it = 0;
            }
            else
                it = it->next;
        }
    }
}

//-----
// Find the object corresponding to the given filename, which is currently
// accessible
// RETURNS: pointer to object, or null
UTL_FileSource * UTL_FileSource::find(const char *filename)
{
    UTL_FileSource * it = head;
    while( it && ! it->isOne(filename) )
        it = it->next;
    if( it && ! it->isAccessible() )
        it = 0;
    return it;
}

//-----
// Determine whether the given filename refers to this object.
// The base class function does a case-insensitive comparison of the given
// filename with the basic name of the object.
// Override this function for classes which respond to more than one filename.
UR_BOOLEAN UTL_FileSource::isOne(
    const char * filename // filename to compare
)
{
    const char *p1 = filename;
    const char *p2 = theFileName;
    while( *p1 == '\\' || *p1 == '/' )
        p1++; // skip leading slashes, so filename is relative to root directory
    while( *p1 && *p2 )
    {
        if( ! filenameCharsMatch(*p1++,*p2++) )
            return UR_FALSE;
    }
    return ( *p1 || *p2 ) ? UR_FALSE : UR_TRUE;
}

//-----
// Extract the title of the file.
// Base class prints the filename.
void UTL_FileSource::printTitle(
    UTL_FileUser & dest, // destination object
    int optionCount,
    const char *options[],
    const char *filename
)
{
    (void)optionCount;
    (void)options;
    (void)filename;
    dest.puts(theFileName);
}

//-----
// Compare two filename characters, to see if they match.
// Case is ignored, and backslash equals slash.
// RETURNS: UR_TRUE if the characters are equivalent, else UR_FALSE
UR_BOOLEAN UTL_FileSource::filenameCharsMatch(
    char c1, // first character
    char c2 // second character
)
{
    if( ( toupper(c1) == toupper(c2) )
        || ( c1=='\\'&&c2=='/' ) )

```

09605010-062700

```

    || ( c2=='\\' && c1=='/' )
        return UR_TRUE;
    else
        return UR_FALSE;
}

//-----
// Checks a file name to see if it matches the pattern baseName###.baseExt and extracts
// the number. Case is ignored, and backslash equals slash.
// RETURNS: UR_TRUE if there is a match.
UR_BOOLEAN UTL_FileSource::filenameMatch
(
    const char * basename, // A base file name.
    const char * filename, // A file name to check.
    unsigned & number      // Place to store the number.
)
{
    UR_BOOLEAN result = UR_TRUE; // Final return value.
    const char * b = basename; // Pointer to base file name.
    const char * p = filename; // Pointer to file name under test.
    unsigned n = 0; // Number of digits collected.
    char digits[16]; // Numeric digits collected from name.

    // Skip leading slashes, so filename is relative to root directory.
    while( *p == '\\' || *p == '/' )
        p++;

    // Compare all the characters up to the dot or null
    while( *b && *b != '.' && result )
    {
        if( ! filenameCharsMatch(*p++,*b++) )
            result = UR_FALSE;
    }

    // If OK so far, gather digits from the given name
    while( *p && (*p != '.') && n < sizeof(digits)-1 && result )
    {
        if( (*p < '0') || (*p > '9') )
            result = UR_FALSE;
        else
            digits[n++] = *p++;
    }

    // If still OK, see if the extension matches
    while( *b && result )
    {
        if( ! filenameCharsMatch(*p++,*b++) )
            result = UR_FALSE;
    }

    // If any digits were collected then convert to a binary number
    // otherwise no match is found.
    if( n && result )
    {
        digits[n] = 0;
        number = atoi( digits );
    }

    return result;
}

//-----
// Checks whether the file is accessible. The file is not accessible if it
// requires a password which has not been entered. It is also not accessible
// if its menu file, or any menu file in the chain, is inaccessible (i.e., each
// file inherits the accessibility of its menu structure).
// RETURNS: UR_TRUE if the file's access requirements are met
UR_BOOLEAN UTL_FileSource::isAccessible(void)
{
    UR_BOOLEAN returnValue = UR_TRUE;
    if( accessLevel == FACTORY_LEVEL )
    {
        if( ! MMI_Application::find()->isFactoryServiceEnabled() )
            returnValue = UR_FALSE;
    }
}

```

09605010 062700


```

if( returnValue && menuFileName && *menuFileName ) // if has a non-blank menu name
{
    UTL_FileSource * menuFile = UTL_FileSource::find(menuFileName);
    if( menuFile )
    {
        if( ! menuFile->isAccessible() )
            returnValue = UR_FALSE; // menu file not accessible, so neither are we
        }
        else
            returnValue = UR_FALSE; // menu doesn't exist, so neither does this file
    }
    return returnValue;
}

//-----
// Delete all UTL_FileSource objects
void UTL_FileSource::deleteAll(void)
{
    while( head )
        delete head;
}

```

Listing 5: UTL_WebPage.h

```

/*****
 * Copyright (C) General Electric Co. GE Confidential and Proprietary
 *
 *****/
#ifndef UTL_WEBPAGE_H
#define UTL_WEBPAGE_H

#include "UTL_FileSource.h"

#define MAX_HTML_TABLE_COLS 40 // maximum number of columns in an HTML table

// =====
// Web page class -- all web pages derive from this class.
// <BR> Key points:
// <UL>
// <LI> Subclasses can override "get", but they shouldn't. The "get" function
// sets up HTTP headers and calls the "printHTML" function, which most
// subclasses also shouldn't override.
// <LI> Subclasses should generally override the "getBody" function, providing
// HTML data for the part of the web page between the start and end
// of the page body.
// <LI> It's not a bad idea to learn some HTML if you're designing pages, but
// you can also use Front Page, Visual Interdev, etc. to design the
// page, then cut-and-paste the HTML into your code.
// <LI> If your page uses tables, use the UTL_WebPage::Table class, it works well.
// <LI> In order to have your web page show up in a menu, specify the
// filename of the UTL_WebMenu object when constructing the page.
// <LI> You may specify an access level when constructing the web page, so
// that, for example, your page shows up only when factory service
// is enabled on the front panel.
// </UL>
// =====
class UTL_WebPage : public UTL_FileSource
{
public:
    UTL_WebPage(const char*filename,const char *aMenuFileName="", DB_SECURITY_LEVEL
anAccessLevel=NO_LEVEL);
    ~UTL_WebPage();
    virtual void get( UTL_FileUser & dest,int optionCount, const char *options[],const
char *filename);
    virtual void printHTML(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);
    virtual void printPageHeading(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);
protected:

    // =====
    // HTML Table class, for use in the getBody function of a UTL_WebPage
    // =====

```

09605010-062700

```

// subclass. Create one on the stack, and it will automatically
// wrap up with the appropriate HTML commands when it de-scopes.
// You can also terminate a table with "end()" in order to start
// a new one using the same object (perhaps calling "setWidth" to
// change the width).
// <BR>
// The startCell, startHeadingCell and startBannerCell functions start
// different types of table cells. The class tracks column usage so
// it will start new rows as required.
// <BR>
// The remaining public functions set font styles. Font styles are valid
// for the remainder of the current cell, and are in general only changed
// right after starting a cell, so the entire contents of the cell have the
// same font.
// =====
// class Table
// {
// public:
//     Table( int aNumCols, UTL_FileUser & dest );
//     ~Table();
//     void setWidth( int aPercentWidth );
//     void startTable(int aNumCols=0);
//     void end(void);
//     void startCell(const char * align="center", const char * bgColor=0, int colspan=1,
// int rowspan=1);
//     void startHeadingCell(const char * align="center");
//     void startBannerCell(void);
//     void setFontStyle(int size=3, const char *color="black");
//     void setFontBold(UR_BOOLEAN onOff = UR_TRUE);
//     void setFontItalic(UR_BOOLEAN onOff = UR_TRUE);
// private:
//     void nextRow(void);
//     void endFont(void);
//     UR_BOOLEAN inTable; // true if between start and end of table
//     UR_BOOLEAN inRow;   // true if between start and end of row
//     UR_BOOLEAN inColumn; // true if between start and end of column
//     UR_BOOLEAN inFont;   // true if between start and end of font definition
//     UR_BOOLEAN isBold;   // true if showing bold text
//     UR_BOOLEAN isItalic; // true if showing italic text
//     int numCols;         // number of columns in the table
//     const char * bgcolor; // background colour
//     const char * fontcolor; // font colour
//     UTL_FileUser & dest;   // !e1725 destination for output
//     int percentWidth;     // width of table, in percent
//     unsigned short usedCols[MAX_HTML_TABLE_COLS]; // to record pre-allocated columns
// for multi-row cells
//     int columnNumber;     // number of column currently being shown (-1 if none yet)
// };

// void linefeed(UTL_FileUser & dest);

// virtual const char * getBackgroundColor(void);
// virtual void getHeader(UTL_FileUser & dest,int optionCount, const char
*options[]);
// // Get the html body text -- sub-classes must define this function.
// virtual void getBody(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename) = 0;
// virtual void printTitle(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);

// // Convert Futaba character set to ISO for web browser.
// // RETURNS: dest, so you can use it in "printf"
// char * webString( char*dest, const char*src );
// };

// =====
// Web page class for menus, which allow users to pick other web pages
// from a list. The "get" function prints a standard layout, including
// all the titles for the web pages which specified the particular
// menu name.
// =====
class UTL_WebMenu : public UTL_WebPage
{
// public:
//     UTL_WebMenu(const char*filename,const char *aMenuFileName="",const char *aTitle=0,
DB_SECURITY_LEVEL anAccessLevel=NO_LEVEL);

```

002290-01050600

```

        virtual void printTitle(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);
        virtual void getBody(UTL_FileUser & dest,int optionCount, const char
*options[],const char *filename);
private:
    const char *title; // menu title
};

#endif

```

Listing 6: UTL_WebPage.cpp

```

/*****
 * Copyright (C) General Electric Co. GE Confidential and Proprietary
 *
 *****/

#include "UTL_WebPage.h"
#include "UTL_FileUser.h"
#include "UTL_StaticFile.h"
#include "DB_Text.h"
#include "DB_IPAddress.h"
#include "DB_UINT16.h"
#include "SYS_Product.h"
#include <stdio.h>
#include <assert.h>

// These data items are used in the standard page banner
extern DB_Text Relay_Name;
extern DB_UINT16 Product_Version;
extern DB_IPAddress IP_Address;

// GIF file with the GE Power Management logo, used in the standard page banner
const char powerManagementLogo[] = {
    "\x47\x49\x46\x38\x39\x61\x69\x00\x5B\x00\xF7\x00\x00\xFF\xFF\xFF\xFF\xFF\xCC\xFF\xFF"
    "\x99\xFF\xFF\x66\xFF\xFF\x33\xFF\xFF\x00\xFF\xCC\xFF\xFF\xCC\xCC\xFF\xCC\x99\xFF\xCC\x66"
    "\xFF\xCC\x33\xFF\xCC\x00\xFF\x99\xFF\xFF\x99\xCC\xFF\x99\x99\xFF\x99\x66\xFF\x99\x33\xFF"
    "\x99\x00\xFF\x66\xFF\xFF\x66\xCC\xFF\x66\x99\xFF\x66\x66\xFF\x66\x33\xFF\x66\x00\xFF\x33"
    "\xFF\xFF\x33\xCC\xFF\x33\x99\xFF\x33\x66\xFF\x33\x33\xFF\x33\x00\xFF\x00\xFF\xFF\x00\xCC"
    "\xFF\x00\x99\xFF\x00\x66\xFF\x00\x33\xFF\x00\x00\xCC\xFF\xFF\xCC\xFF\xCC\xCC\xFF\x99\xCC"
    "\xFF\x66\xCC\xFF\x33\xCC\xFF\x00\xCC\xCC\xFF\xCC\xCC\xCC\xCC\xCC\x99\xCC\xCC\x66\xCC\xCC"
    "\x33\xCC\xCC\x00\xCC\x99\xFF\xCC\x99\xCC\xCC\x99\x99\xCC\x99\x66\xCC\x99\x33\xCC\x99\x00"
    "\xCC\x66\xFF\xCC\x66\xCC\xCC\x66\x99\xCC\x66\x66\xCC\x66\x33\xCC\x66\x00\xCC\x33\xFF\xCC"
    "\x33\xCC\xCC\x33\x99\xCC\x33\x66\xCC\x33\x33\xCC\x33\x00\xCC\x00\xFF\xCC\x00\xCC\xCC\x00"
    "\x99\xCC\x00\x66\xCC\x00\x33\xCC\x00\x00\x99\xFF\xFF\x99\xFF\xCC\x99\xFF\x99\x99\xFF\x66"
    "\x99\xFF\x33\x99\xFF\x00\x99\xCC\xFF\x99\xCC\xCC\x99\xCC\x99\x99\xCC\x66\x99\xCC\x33\x99"
    "\xCC\x00\x99\x99\xFF\x99\x99\xCC\x99\x99\x99\x99\x99\x66\x99\x99\x33\x99\x99\x00\x99\x66"
    "\xFF\x99\x66\xCC\x99\x66\x99\x99\x66\x66\x99\x66\x33\x99\x66\x00\x99\x33\xFF\x99\x33\xCC"
    "\x99\x33\x99\x99\x33\x66\x99\x33\x33\x99\x33\x00\x99\x00\xFF\x99\x00\xCC\x99\x00\x99\x99"
    "\x00\x66\x99\x00\x33\x99\x00\x00\x66\xFF\xFF\x66\xFF\xCC\x66\xFF\x99\x66\xFF\x66\x66\xFF"
    "\x33\x66\xFF\x00\x66\xCC\xFF\x66\xCC\xCC\x66\xCC\x99\x66\xCC\x66\x66\xCC\x33\x66\xCC\x00"
    "\x66\x99\xFF\x66\x99\xCC\x66\x99\x99\x66\x99\x66\x66\x99\x33\x66\x99\x00\x66\x66\xFF\x66"
    "\x66\xCC\x66\x66\x99\x66\x66\x66\x66\x66\x33\x66\x66\x00\x66\x33\xFF\x66\x33\xCC\x66\x33"
    "\x99\x66\x33\x66\x66\x33\x33\x66\x33\x00\x66\x00\xFF\x66\x00\xCC\x66\x00\x99\x66\x00\x66"
    "\x66\x00\x33\x66\x00\x00\x33\xFF\xFF\x33\xFF\xCC\x33\xFF\x99\x33\xFF\x66\x33\xFF\x33\x33"

```

09605010 "062700


```
"\x28\x90\x14\x0E\xFE\x26\xE1\x84\x14\x56\x68\xE1\x85\x18\x66\xA8\xE1\x1C\x76\xE8\xE1"
"\x87\x20\x86\x28\xE2\x88\x24\x96\x68\xE2\x89\x28\xA6\xA8\xE2\x8A\x2C\x4A\x64\x5C\x7A\xCE"
"\x1D\x55\x58\x60\x7C\xE9\x75\x17\x6C\x2B\x01\xB8\x90\x4A\xD0\xE5\xE7\x93\x41\xD4\x59\x76"
"\x58\x8F\x0E\x3D\x96\x10\x73\x08\xAD\x44\xE4\x59\xDF\x3D\xE4\x1E\x42\xF0\x0D\x18\xD2\x5A"
"\x43\x0E\x25\xD8\x41\x74\x39\x98\xE3\x5A\xFA\x31\xF4\x24\x55\xF8\x25\x29\x5F\x5A\x48\x46"
"\x84\x58\x62\x04\x5E\xE5\x56\x7F\x60\x45\x39\x98\x7D\xC3\xED\x25\xE7\x00\x6C\x8A\xD5\xA4"
"\x73\x05\xE8\x38\x90\x8C\x03\x65\x19\x5B\x95\x11\x19\x19\x67\x48\xD9\x75\x69\xA0\x74\x4E"
"\x2E\x78\x90\x7E\x80\x7E\xA6\x18\x44\xCF\x41\x26\xE0\x66\x69\x7E\x46\xDF\x43\x2E\x45\xF5"
"\xA8\x8F\x53\x7E\x86\x55\x84\xF1\xC1\x39\x17\xA2\x06\x35\x6A\x19\x9F\xF7\xE9\x89\x9C\xA1"
"\x7B\x8E\xB9\x5C\x5D\x26\x76\x1D\x75\x23\x94\xE7\xCD\xE9\x17\x98\x70\xDD\x1A\x5B\x00\xBC"
"\xD6\xD9\xE2\xAF\xC0\x06\x2B\xEC\xB0\xC4\x16\x6B\xEC\xB1\xC8\x26\xAB\xEC\xB2\xB8\x05\x04"
"\x00\x3B"
};
static UTL_StaticFile * urGifFile = 0;

// The default web page (main menu)
static UTL_WebMenu * mainMenuFile = 0;

//=====
// Constructor -- creates a web page
//=====
UTL_WebPage::UTL_WebPage(
    const char*filename, // filename by which to find this file
    const char *aMenuFileName, // filename of menu in which to
include this file
    DB_SECURITY_LEVEL anAccessLevel // access level
)
: UTL_FileSource(filename,anAccessLevel)
{
    menuFileName = aMenuFileName;
    if( !logoFile ) // Use logoFile pointer to see if the whole group has been constructed
    {
        logoFile = new UTL_StaticFile("/logo.gif", (unsigned char*)&powerManagementLogo,
sizeof(powerManagementLogo));
        assert( logoFile );
        urGifFile = new UTL_StaticFile("/URWellConnected.gif", (unsigned
char*)&URWellConnected, sizeof(URWellConnected));
        assert( urGifFile );
        mainMenuFile = new UTL_WebMenu("/default.htm","", "Main Menu");
        assert( mainMenuFile );
    }
}

//=====
// Destructor
//=====
UTL_WebPage::~UTL_WebPage()
{
}

//=====
// Get the contents of the web page, including HTML header info.
//=====
void UTL_WebPage::get(
    UTL_FileUser & dest, // where to send the data
    int optionCount, // number of options
    const char *options[], // options -- ignored at this level, but may be used in
other functions
    const char *filename // filename - ignored, since web pages use options instead
)
{
    (void)filename;
    // Format the header part
    dest.puts( (char*)
        "HTTP/1.0 200 OK \r\n"
```

```

    "Server: GE Industrial Systems UR\r\n"
    "Content-Type: text/html\r\n"
    "\r\n");
    printHTML( dest, optionCount, options, filename );
}

//=====
// Print the non-header part of the page. Override if you don't want the normal
// head, body layout (for a frameset, for example).
//=====
void UTL_WebPage::printHTML(
    UTL_FileUser & dest,          // where to send the file
    int optionCount,             // number of options
    const char *options[],        // options
    const char *filename         // filename, in case it matters
)
{
    dest.puts(
        "<HTML>\n"
        "<HEAD>\n"
        "<TITLE>" );
    char s[100], a[100];
    UR_UINT16 c;
    (void)Relay_Name.get((char*)s);
    (void)Relay_Name.toAscii(&c,(char*)a,(void*)s);
    dest.puts(a);
    dest.puts(" ");
    printTitle(dest,optionCount,options,filename);
    dest.puts( "</TITLE>\n");
    getHeader(dest,optionCount,options);
    dest.puts( "</HEAD>\n" );
    // format the body
    dest.printf( "<BODY BGCOLOR=%f.50s>\n", getBackgroundColor());
    printPageHeading( dest, optionCount, options, filename );
    getBody(dest,optionCount,options,filename);
    dest.puts( "</BODY></HTML>\n" );
}

//=====
// Print a banner or heading at the top of the web page. Override if you don't
// want the normal heading (unless you're already overriding printHTML, from which
// this function is called).
//=====
void UTL_WebPage::printPageHeading(
    UTL_FileUser & dest,          // where to send the file
    int optionCount,             // number of options
    const char *options[],        // options
    const char *filename         // filename, in case it matters
)
{
    (void)optionCount;
    (void)options;
    (void)filename;
    UR_UINT16 count;           // Number of characters printed to buffer.
    char buffer[256];         // Buffer for use with getFormattedLine.
    UR_UINT16 versionNumber;

    UR_BOOLEAN isHomePage = isOne("default.htm");

    Table t(4,dest);
    t.startCell("left","black",1,2);
    dest.puts(
        "<IMG SRC=\"/logo.gif\" ALT=\"GE Power Management Logo\" TITLE=\"GE Power
Management Logo\">\n"
    );

    t.startCell("left","silver");
    t.setFontBold();

    dest.printf("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~<br>", SYS_Product::find()->name );
    (void)Product_Version.get(&versionNumber);
    (void)Product_Version.toAscii(&count,(char*)&buffer,&versionNumber);
    dest.printf("&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~&nbsp;&nbsp;&nbsp;&nbsp;&~&nbsp;&nbsp;&~&nbsp;&~&~&~&~&~&~&~&~&~&~&~&~&~&~&~&~&~&~&s", buffer);

    t.startCell("right","silver");
    t.setFontItalic();

```



```

// Convert Futaba character set to ISO for web browser.
// RETURNS: dest, so you can use it in "printf"
//=====
char * UTL_WebPage::webString(
                                char*dest,          // destination buffer -- make sure it's
big enough                                const char*src    // source string
                                )
{
    char *p = dest;
    while( *src )
    {
        switch( 255 & (*src) )
        {
            case 0x7f: // all pixels on
                p += sprintf(p,"%Xi;");    // not perfect, but I guess it will do
                break;
            case 0xDF: // degree
                p += sprintf(p,"%deg;");
                break;
            case 0x88: // micro
                p += sprintf(p,"%mu;");
                break;
            case 0x8e: // ohms
                p += sprintf(p,"%Omega;");
                break;
            case 0x8d: // phase symbol
                p += sprintf(p,"%Phi;");
                break;
            default:
                *p++ = *src;
                break;
        }
        src++;
    }
    *p = 0;
    return dest;
}

//=====
// Table constructor -- creates an HTML table, which will terminate on de-scoping.
// You should generally create this guy on the stack.
// EXAMPLE:
//     SomeSubclass::getBody( ...
//     {
//         ...
//         UTL_WebPage::Table t(2,dest);
//
//         t.startBannerCell();
//         printTitle(dest,optionCount,options,filename);
//         t.nextRow();
//         t.startHeadingCell();
//         dest.puts("first column heading");
//         t.startHeadingCell("left");
//         dest.puts("second column heading");
//
//         while( some condition )
//         {
//             (get_next_row_data)
//             t.nextRow();
//             t.startCell();
//             dest.printf("%d", some_value);
//             t.startCell("left");
//             dest.printf("%s", some_text);
//         }
//         ...
//     }
//=====
UTL_WebPage::Table::Table(
                            int aNumCols,          // number of table columns
                            UTL_FileUser & aDest  // destination for the HTML output
                            )
: dest(aDest)
{
    inTable = UR_FALSE;
    inRow = UR_FALSE;
}

```

```

inColumn = UR_FALSE;
inFont = UR_FALSE;
isBold = UR_FALSE;
isItalic = UR_FALSE;
numCols = aNumCols > MAX_HTML_TABLE_COLS ? MAX_HTML_TABLE_COLS : aNumCols;
bgcolor = 0;
fontcolor = 0;
percentWidth = 95;
columnNumber = -1;
for( int i=0; i<MAX_HTML_TABLE_COLS; i++ )
    usedCols[i] = 0;
}

//=====
// Table destructor -- terminates HTML table, if one has started.
//=====
UTL_WebPage::Table::~Table()
{
    end();
    bgcolor = 0;
    fontcolor = 0;
}

//=====
// Set the width of the table, in percent. Call this function before any of the
// other functions, to set the width different from the default (95%).
//=====
void UTL_WebPage::Table::setWidth(
    int aPercentWidth // width of subsequently-started table (5-100 percent)
)
{
    percentWidth = aPercentWidth;
    if( percentWidth > 100 )
        percentWidth = 100;
    if( percentWidth < 5 )
        percentWidth = 5;
}

//=====
// Start a table, terminating the previous one if it's started.
//=====
void UTL_WebPage::Table::startTable(
    int aNumCols // number of table columns
)
{
    end();
    if( aNumCols > 0 )
        numCols = aNumCols > MAX_HTML_TABLE_COLS ? MAX_HTML_TABLE_COLS : aNumCols;

    dest.puts("<BR>\r\n");
    dest.printf("<TABLE width=%d%% align=center bgColor=#F0F0F0 border=2 borderColor=black
border=2 cellspacing=0 cellpadding=3>",
        percentWidth);
    inTable = UR_TRUE;
    columnNumber = -1;
    for( int i=0; i<MAX_HTML_TABLE_COLS; i++ )
        usedCols[i] = 0;
}

//=====
// Terminate the table -- generally only call this function if you want to output
// some HTML before starting another table (otherwise you can rely on the destructor).
//=====
void UTL_WebPage::Table::end(void)
{
    endFont();
    if( inColumn )
    {
        dest.puts("</TD>");
        inColumn = UR_FALSE;
    }
    if( inRow )
    {
        dest.puts("</TR>");
        inRow = UR_FALSE;
    }
}

```

```

if( inTable )
{
    dest.puts("</TABLE>");
    inTable = UR_FALSE;
}
}

//=====
// Start a row of cells, wrapping up the previous row, if any, and starting
// a table, if not already started.
//=====
void UTL_WebPage::Table::nextRow(void)
{
    endFont();
    if( !inTable )
        startTable();
    if( inColumn )
    {
        dest.puts("</TD>");
        inColumn = UR_FALSE;
    }
    if( inRow )
        dest.puts("</TR>");

    dest.puts("\r\n<TR valign=center>");
    inRow = UR_TRUE;
    columnNumber = -1;
}

//=====
// Start a column, wrapping up the previous one if any, and starting the table
// and/or row if necessary.
//=====
void UTL_WebPage::Table::startCell(
    const char * align,      // alignment ("center","left","right")
    const char * bgColor,    // background colour ("white","silver","yellow", etc.)
    int colspan,             // number of columns to span (generally 1)
    int rowspan              // number of rows to span (generally 1)
)
{
    int i;

    endFont();

    if( !inTable )
        startTable();
    if( ! inRow )
        nextRow();
    if( inColumn )
        dest.puts("</TD>");

    // Find the columns which can hold our cell.
    // If this is a multi-row cell, reserve the columns it needs.
    // Expect screw-ups if the configuration is truly whacky, like a colspan cell
    // spanning over a previous-row rowspan cell.
    int colsToReserve = colspan;
    if( (columnNumber+1) >= numCols )    // if last row ended at end of row...
        nextRow();                      // ...start a new row
    while( colsToReserve )
    {
        if( ++columnNumber < numCols )
        {
            if( usedCols[columnNumber] )
                usedCols[columnNumber]--;    // can't use this one, but absorb a
reservation
            else
            {
                if( rowspan > 1 )
                    usedCols[columnNumber] = rowspan-1;    // reserve the column for as
many rows as necessary
                colsToReserve--;
            }
        }
        else
        {
            nextRow();    // end of the row

```

```

        colsToReserve = 0; // get out of here
    }
}

if( rowspan > 1 )
{
    short minReserved = 32767;
    for( i=0; i<numCols; i++ )
    {
        if( usedCols[i] < minReserved )
            minReserved = usedCols[i];
    }
    if( minReserved )
    {
        for( i=0; i<numCols; i++ )
            usedCols[i] -= minReserved; // eliminate completely reserved rows
    }
}

inColumn = UR_TRUE;
dest.puts("<TD");

if( colspan > 1 )
    dest.printf(" colspan=%d", colspan);
if( rowspan > 1 )
    dest.printf(" rowspan=%d", rowspan);
if( bgColor )
    dest.printf(" bgcolor=%s", bgColor);
dest.printf(" align=%s>", align);
}

//=====
// Start a "heading" cell, with special highlight formatting
//=====
void UTL_WebPage::Table::startHeadingCell(const char * align)
{
    startCell(align,"silver");
    dest.puts("<FONT color=black size=4><STRONG>\r\n");
    isBold = UR_TRUE;
    inFont = UR_TRUE;
}

//=====
// Start a "banner" cell, spanning an entire row, with special highlighting.
//=====
void UTL_WebPage::Table::startBannerCell(void)
{
    endFont();

    if( !inTable )
        startTable();
    if( inColumn )
        dest.puts("</TD>");
    if( inRow )
        dest.puts("</TR>");

    dest.printf("\r\n<TR><TD align=center colspan=%d bgcolor=#483D8B><FONT color=white
size=5><STRONG>",numCols);
    columnNumber = numCols; // force next cell to new row
    inRow = UR_TRUE;
    inColumn = UR_TRUE;
    inFont = UR_TRUE;
    isBold = UR_TRUE;
}

//=====
// Turn off any special font formatting.
//=====
void UTL_WebPage::Table::endFont(void)
{
    if( isItalic )
    {
        isItalic = UR_FALSE;
        dest.puts("</EM>");
    }
    if( isBold )
    {

```

09605010-062700

```

        isBold = UR_FALSE;
        dest.puts("</STRONG>");
    }
    if( inFont )
    {
        inFont = UR_FALSE;
        dest.puts("</FONT>");
    }
}

//=====
// Change font style for remainder of this table cell
//=====
void UTL_WebPage::Table::setFontStyle(
    int size,          // size of font (normal is 3)
    const char *color  // font colour
)
{
    if( inFont )
        dest.puts("</FONT>");
    inFont = UR_TRUE;
    dest.printf("<FONT size=%d color=%s>", size, color );
}

//=====
// Turn bold text on or off for remainder of this table cell
//=====
void UTL_WebPage::Table::setFontBold(
    UR_BOOLEAN onOff   // true for bold, false for noraml
)
{
    if( onOff )
    {
        if( !isBold )
            dest.puts("<STRONG>");
        isBold = UR_TRUE;
    }
    else
    {
        if( isBold )
            dest.puts("</STRONG>");
        isBold = UR_FALSE;
    }
}

//=====
// Turn italic text on or off for remainder of this table cell
//=====
void UTL_WebPage::Table::setFontItalic(
    UR_BOOLEAN onOff   // true for italic, false for noraml
)
{
    if( onOff )
    {
        if( !isItalic )
            dest.puts("<EM>");
        isItalic = UR_TRUE;
    }
    else
    {
        if( isItalic )
            dest.puts("</EM>");
        isItalic = UR_FALSE;
    }
}

//=====
// Constructor -- creates a web page for a menu
//=====
UTL_WebMenu::UTL_WebMenu(
    const char*filename,          // filename by which to find this file
    const char *aMenuFileName, // filename of menu in which to include
    this file
    const char *aTitle,          // menu title
    DB_SECURITY_LEVEL anAccessLevel // access level

```

09605010-062700


```

// =====
// Generic file user class, to obtain data from UTL FileSource objects.
// Subclasses override the sendFrame function to modify the mechanics
// involved in getting blocks of data where they have to go.
// <BR> Key functions are:
// <UL>
// <LI> printf - formatted, buffered print
// <LI> puts - buffered write of a string
// <LI> write - block write
// <LI> flush - send any unsent information from the buffers
// </UL>
// =====
class UTL_FileUser
{
public:
    int printf( const char * fmt, ...); //lint !e1916
    void puts( const char * txt );
    void write(unsigned char *buffer, UR_UINT16 length);
    void flush(void);
    UR_UINT16 getLength(void)      // get the maximum buffer length
    {
        return theLength;
    }
    unsigned char * getBuffer(void)      // get a buffer into which to format the
frames
    {
        return theBuffer;
    }
    virtual ~UTL_FileUser();
protected:
    virtual void sendFrame(unsigned char *buffer, UR_UINT16 length) = 0;
    UTL_FileUser(unsigned char *buffer, UR_UINT16 length);
    unsigned char * theBuffer;      // points to a handy buffer for formatting messages
    UR_UINT16 theLength;            // size of the handy buffer
    UR_UINT16 bufferedChars;        // number of characters waiting to be sent
};

#endif

```

Listing 8: UTL_FileUser.cpp

```

/*****
 * Copyright (C) General Electric Co. GE Confidential and Proprietary
 *
 * DESCRIPTION File user class
 *
 *****/

#include "UTL_FileUser.h"
#include <assert.h>

// va_list is defined differently in visual C++ and GNU, so we need to tweak the
// code to match the compiler being used.
#ifdef WIN32
    // Definitions from the Visual C++ stdarg.h
    #undef va_start
    #undef va_end
    #define va_start(ap,v) (ap = (char*)&v + ( (sizeof(v) + sizeof(int) - 1) &
~(sizeof(int) - 1) ))
    #define va_end(ap)      ( ap = (char*)0 )
    extern int TARGET_VSNPRINTF(char*,size_t,const char*,char*);
#else
    #ifndef _lint
        #include <stdarg.h> // skip this GNU header for win32
        #include <stdio.h>
    #endif
    #define TARGET_VSNPRINTF(a,b,c,d) vsprintf(a,c,d)
#endif

// Like stdio.h printf, but writes to UTL_FileSource
// *WARNING* Don't write too much data -- 500 chars max!
int UTL_FileUser::printf( const char * fmt, ... ) //lint !e1916
{
#ifdef _lint

```



```

        assert(1); // to make lint happy
        return 0; // the real function is just to crazy for lint
    #else
    #ifdef WIN32
        // definition from Visual C++ stdio.h, with defines resolved (different from GNU)
        #define target_va_list char*
    #else
        #define target_va_list va_list
    #endif
    char tmp[500];
    target_va_list ap;
    va_start(ap, fmt);
    int ret = TARGET_VSNPRINTF(tmp, sizeof(tmp), fmt, ap);
    assert( ret >= 0 && ret <= (int)(sizeof(tmp)) );
    va_end (ap);
    puts((const char *)tmp);
    return (ret);
    #endif
}

// write a null-terminated string, with buffering
void UTL_FileUser::puts( const char * txt )
{
    const char * p = txt;
    while( *p )
    {
        if( bufferedChars >= theLength )
            flush();
        theBuffer[bufferedChars++] = (unsigned char)*p++;
    }
}

// write data
void UTL_FileUser::write(unsigned char *buffer, UR_UINT16 length)
{
    flush();
    sendFrame(buffer, length);
}

// Ensure that all data has been sent
void UTL_FileUser::flush(void)
{
    if( bufferedChars )
    {
        sendFrame(theBuffer, bufferedChars );
        bufferedChars = 0;
    }
}

UTL_FileUser::UTL_FileUser(unsigned char *buffer, UR_UINT16 length)
{
    theBuffer = buffer;
    theLength = length;
    bufferedChars = 0;
}

UTL_FileUser::~~UTL_FileUser()
{
}

```